(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

SC13158TH AE

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
24 April 2003 (24.04.2003)

PCT

(10) International Publication Number
WO 03/034225 A2

(51) International Patent Classification[7]: G06F 11/36

(21) International Application Number: PCT/GB02/04548

(22) International Filing Date: 7 October 2002 (07.10.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
0124554.7 12 October 2001 (12.10.2001) GB
60/337,314 6 December 2001 (06.12.2001) US

(71) Applicant *(for all designated States except US)*: **SIROYAN LIMITED** [GB/GB]; 12-15 Hanger Green Lane, London W5 3AY (GB).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **EDWARDS, David, Alan** [GB/GB]; 6 Pro-Cathedral Lane, Clifton, Bristol BS8 1LB (GB). **JACKSON, Jason, Mark** [GB/GB]; Flat 1, 72A Parchment Street, Winchester, Hampshire SO23 8AT (GB).

(74) Agent: **HITCHING, Peter, Matthew**; Haseltine Lake, 15-19 Kingsway, London WC2B 6UD (GB).

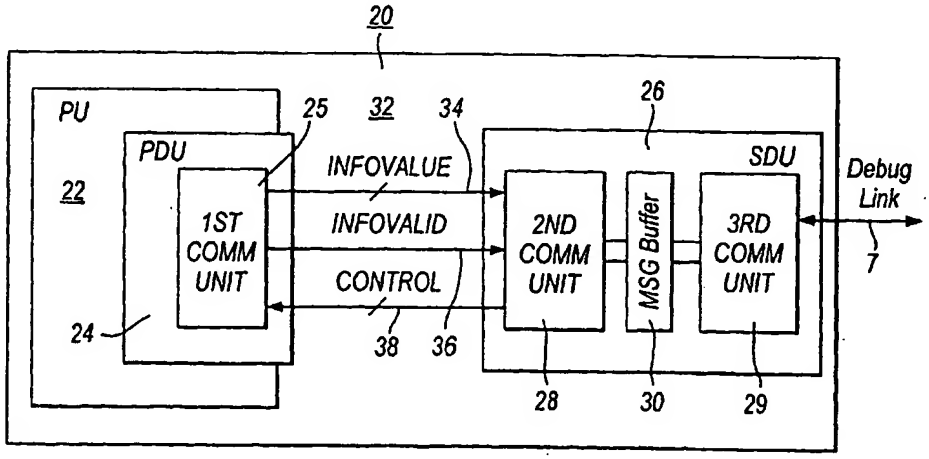
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, SK, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

[Continued on next page]

(54) Title: DEBUGGING OF PROCESSORS



(57) Abstract: Processor circuitry such as a "system-on-chip" (SOC) device comprises a processing unit (22) and a processor debug unit (24) which generates one or more debug information signals for the processing unit. A system debug unit (26) is connected with the processor debug unit by a signal path (34), and receives each debug information signal and outputs one or more further signals derived therefrom to further circuitry (7). Message transfer circuitry (25, 28, 29, 30) transfers one or more messages, each including at least one debug information signal, from the processor debug unit to the system debug unit via the signal path. At least one said message has a number of bits greater than a width of the signal path (34), and the message transfer circuitry transfers that message from the processor debug unit to said system debug unit in a series of message cycles. Such processor circuitry can enable a rich set of debug information signals to be transferred from the processor debug unit to the system debug unit without extensive interconnect between them. More and more one processor debug unit may be supported by the same system debug unit.

WO 03/034225 A2

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

DEBUGGING OF PROCESSORS

The present invention relates to debugging of processors. In particular, but not exclusively, the present invention relates to debugging of processor systems having one or more individual processing units connected to a common system debug module or unit. Such a processor system may be a system-on-chip (SOC) device.

Fig. 1 is a schematic view of a debugging system suitable for debugging a processor core (CPU core) 2 in an SOC device 1. The aim of the debugging process is to identify and correct any problems associated with programs executed by the processor core 2. The debugging system comprises a debug module 3 forming part of the SOC device 1, a debug adaptor 4 and a host computer 5 running debugger software 6. The debug module 3 and debug adaptor 4 are able to communicate with one another via a debug link 7. The debug adaptor 4 can provide debug control signals to the debug module 3 via the debug link 7 and can receive debug information generated by the debug module 3 via the debug link 7. The debug adaptor 4 and host computer 5 are able to communicate with one another via a data link 8, which is also a bi-directional link such as an Ethernet connection or a PCI bus.

The debug adaptor 4 comprises a combination of hardware and software capable of interfacing between the host computer 5 and the debug module 3, for example translating between different communication protocols used by the links 7 and 8, and buffering debug information received from the debug module 3 prior to onward transmission to the host computer 5.

Although Fig. 1 shows only a single processor core 2 in the SOC device 1, the Fig. 1 system may be used to debug an SOC device (processor system) including a

plurality of processor cores. In this case, the debug module 3 preferably supports all the processor cores, enabling each of them to be linked to the debug adaptor 4 via the debug link 7. Similarly, if the processor core 2 in Fig. 1 has more than one individual processing unit (e.g. a VLIW processor core), the debug module 3 may handle debug information for all the individual processing units.

Incidentally, instead of using a debug adaptor 4 linked to a host computer 5 running debugger software 6 as shown in Fig. 1, it is possible to arrange for the debugger software 6 to be executed by the or each processor core being debugged. Such an arrangement has the advantage of dispensing with the debug adaptor 4 and host computer 5 but suffers from the disadvantage of being very intrusive in the normal execution of the processor core(s).

In the Fig. 1 debugging system several mechanisms may be provided for generating debug information and supplying it to the debugger software 6 running on the host computer 5. These mechanisms include breakpoints, watchpoints, branch trace messaging, CPU event detectors and data acquisition messaging.

A breakpoint can be set in the processor core 2 to raise a debug exception when a specified event occurs during execution of a program by the processor core 2. This debug exception causes a debug exception handler to execute in preference to the normal execution of the program being debugged. The breakpoint may be a data breakpoint or an instruction breakpoint. A data breakpoint triggers a debug exception when a load/store operation is performed in a specified address range. An instruction breakpoint triggers a debug exception when an instruction within a specified address range is executed. Breakpoints are inherently intrusive monitoring mechanisms because they involve suspending

the normal execution of the program being debugged so that the debug exception handler can execute. By slowing or pausing the normal execution, some types of error may not be reproduced, and thus cannot be detected or corrected. This can result in so-called "Heisenbugs" (named after the Heisenberg Uncertainty Principle), i.e. situations in which the presence or operation of the debugging system perturbs the normal operation or the processor core such that the bug is no longer manifested.

A watchpoint is similar to a breakpoint but, instead of raising a debug exception when the specified event occurs, the watchpoint causes some other action to take place, for example generating a signal made available externally of the processor core 2 by the debug module 3 or generating a trace message. For example, a data watchpoint may generate a data trace message when a load/store operation is performed within a specified address range. Similarly, an instruction watchpoint may generate an instruction trace message when an instruction within a specified address range is executed.

Instruction breakpoints/watchpoints (IBWs) allow the debugger software 6 to detect execution of specific lines of source code in the processor core when the breakpoint/watchpoint is set to a single address, or to detect execution of specific functions in the processor core when the instruction breakpoint/watchpoint is set to a range of addresses. Data breakpoints/watchpoints (DBWs) allow the debugger software 6 to detect when the processor core reads or writes to specific source-code variables and/or to specific memory mapped registers.

The processor core 2 can be programmed to monitor non-sequential execution events within the core, for example conditional branches, unconditional branches, subroutine calls and returns, system calls, exceptions

and interrupts. When these events occur a branch trace message (BTM) can be generated by the debug module 3 comprising, for example, the program counter values before and after the branch is taken. Such branch trace messaging allows the debugger software 6 to monitor the control flow of the processor core 2.

The processor core 2 may also include CPU event detectors (CEDs) used to monitor various CPU events such as retiring of an instruction, a cache hit, a cache miss, a translation lookaside buffer (TLB) hit, a TLB miss, a pipeline stall etc. The processor core 2 may also include counters so that each time a particular event is detected by a CED a count value is incremented. The information gathered using CEDs can be used to optimise the processor core and/or the design of the overall system in which it operates.

The processor core 2 may also write data to one or more data acquisition messaging registers provided for the processor core. A data acquisition message (DQM) is supplied to the debug link 7 in response to the writing of such data. The DQM typically contains the program counter value of the instruction which wrote to the DQM register, and the data value which was written to the register. If the processor core has a virtual memory mechanism, the DQM may also contain a current address space number (CASN) which is a value held in a register of the processor core and used by the virtual memory mechanism to select which translation lookaside buffer (TLB) entries are available to the executing instruction stream.

Fig. 2 shows parts of the SOC device 1 of Fig. 1 in more detail. In Fig. 2 the debug circuitry provided within the SOC device is shown divided into two parts, namely a processor debug unit (PDU 12) and a system debug unit (SDU) 14. The PDU 12 comprises logic circuitry implemented within the pipeline logic of the

processor core (processing unit PU) 2 and serves to monitor states caused by the execution of instructions. Processor-based actions, for example the raising of debug-related exceptions, are generated entirely by the PDU logic circuitry itself.

The PDU logic circuitry includes one or more debug event detectors (not shown) used to detect one or more preselected debug events occurring in the processor core. These debug events may be, for example, instruction breakpoints or watchpoints (IBWs), data breakpoints or watchpoints (DBWs), branch events requiring generation of branch trace messaging (BTM), data acquisition messaging events requiring generation of data acquisition messaging (DQM), or other CPU events (cache hit, cache miss, etc). These debug event detectors are implemented within the pipeline logic of the processor core 2 due to timing constraints and a requirement for close integration with pipeline behaviour (to ensure that precise debug exceptions can be generated).

The SDU 14 provides the SOC device 1 with a communication port (sometimes referred to as a AUX port) for connection via a debug link 7 to a debug adaptor 3 as shown in Fig. 1. This permits the debug adaptor 3 to access the SOC device 1. The SDU 14 is connected via interconnection circuitry 18 to the PDU 12 for receiving debug information signals (status and trace signals) therefrom and for supplying control signals thereto. These signals are clocked at the operating frequency of the processor core 2 and are updated for each instruction retired in that processor core 2.

As will be described in more detail later in the present specification, conventionally the interconnection circuitry 18 is required to have a very large number of signal paths (wires) between the PDU 12

and the SDU 14. For example, a paper entitled "MCORE Architecture implements real-time debug port based on NEXUS Consortium specification", David Ruimy Gonzales, presented at the Embedded Systems Chicago conference in March 1999 (www.ieee-isto.org/Nexus/5001/whitepapers.html) describes interconnection circuitry (an MCORE interface) which uses a minimum of 130 wires. Routing this number of wires between the PDU and SDU is problematic, as it can have a significant impact on the available area within the SOC device, as well as on the maximum clock speed possible. Furthermore, having so many wires with signals changing on every clock cycle is not power efficient. An embedded trace macrocell (ETM) available from ARM (www.arm.com) also uses a large number of wires.

These problems are exacerbated in processor systems having more than one processing unit, for example an SOC device having a plurality of processor cores. For example, if the processor system can retire more than one instruction in each cycle, it is not possible to supply debug information signals from the two or more processing units to the system debug unit using the interconnection circuitry. This problem could be solved by duplicating the PDU 12, SDU 14 and interconnection circuitry 18 for each processor core, but this leads to a proliferation of the debug circuitry as a whole.

In view of size constraints on the interconnection circuitry, conventional debug arrangements have also tended to economise on the amount of debug information transmittable from the PDU to the SDU. For example, the MCORE interface and ARM ETM do not transmit the PC value from the PDU in response to a DBW/DQM event.

Another limitation of conventional debug arrangements is that the signals transmitted between

the PDU and the SDU are very specific to the particular processor core, making it necessary to provide different designs of the interconnection circuitry 18 and/or SDU 14 for each type of processor core to be debugged.

Finally, conventional debugging arrangements have tended to have "closely-coupled" PDUs and SDUs with no clear division between them. There are some situations in which it may be desirable to be able to omit the SDU from production devices, for example because the SDU could be used by hackers/counterfeiters to obtain access to restricted information contained within the device or for reducing the amount of circuitry included in the production devices. In the closely-coupled conventional debug arrangements the omission of the SDUs is not readily achievable.

According to a first aspect of the present invention there is provided processor circuitry comprising: a processing unit; a processor debug unit provided for said processing unit and operable to generate one or more debug information signals for said processing unit; a system debug unit, connected with said processor debug unit by a signal path, and operable to receive each said debug information signal and to output one or more further signals derived therefrom to further circuitry; and message transfer means operable to transfer one or more messages, each including at least one said debug information signal, from said processor debug unit to said system debug unit via said signal path, at least one said message having a number of bits greater than a width of said signal path, and said message transfer means being operable to transfer that message from said processor debug unit to said system debug unit in a series of message cycles.

According to a second aspect of the present

invention there is provided processor circuitry comprising: a processing unit; a processor debug unit provided for said processing unit and operable to generate one or more debug information signals for said processing unit; wherein said processor debug unit comprises a communication unit having a signal path output adapted for connection to a signal path having a preselected width, and operable to output from said signal path output one or more messages, each including at least one said debug information signal, at least one said message having a number of bits greater than said preselected width, and said communication unit being operable to output that message from said signal path output in a series of message cycles.

The width of a signal path generally means the number of wires available to transmit independent bits in parallel. In some situations a signal path may provide more than one wire per bit, for example two wires for transmitting complementary versions of the bit. In this case, the effective width of the signal path is still limited to the number of independent bits that can be transmitted in parallel.

Not all messages in the first and second aspects of the invention need require two or more message cycles. At least one message may be transferred in a single message cycle.

In the second aspect of the invention the system debug unit is omitted from the processor circuitry, as may be desired in a production device.

According to a third aspect of the present invention there is provided processor circuitry comprising: a plurality of processing units; a plurality of processor debug units each provided for a corresponding one of said processing units and operable to generate one or more debug information signals for its said corresponding processing unit; a system debug

unit, connected with each of said processor debug units by interconnection means providing a signal path from each processor debug unit to the system debug unit, and operable to receive each said debug information signal generated by the plurality of the processor debug units and to output one or more further signals derived therefrom to further circüitry; and message transfer means operable to transfer one or more messages, each including at least one said debug information signal, from each said processor debug unit to said system debug unit via that processor debug unit's said signal path to the system debug unit, messages from different processor debug units being transferred to the system debug unit at different respective times.

According to a fourth aspect of the present invention there is provided processor circuitry comprising: a plurality of processing units; a plurality of processor debug units each provided for a corresponding one of said processing units and operable to generate one or more debug information signals for its said corresponding processing unit; a system debug unit, connected with at least one said processor debug unit by a signal path, and operable to receive each said debug information signal generated by the or each connected processor debug unit and to output one or more further signals derived therefrom to further circuitry; and message transfer means operable to transfer one or more messages, each including at least one said debug information signal, from the or each said connected processor debug unit to said system debug unit via said signal path; wherein: at least one said debug information signal available in said first processing unit is not available in said second processing unit; and said processor debug unit corresponding to said second processing unit produces a dummy information signal which is included in place of

said unavailable debug information signal in one of said messages sent by that processor debug unit to said system debug unit.

According to a fifth aspect of the present invention there is provided processor circuitry comprising: at least first and second processing units; a plurality of processor debug units each provided for a corresponding one of said processing units and operable to generate one or more debug information signals for its said corresponding processing unit; a system debug unit, connected with at least one said processor debug unit by a signal path, and operable to receive each said debug information signal generated by the or each connected processor debug unit and to output one or more further signals derived therefrom to further circuitry; and message transfer means operable to transfer one or more messages, each including at least one said debug information signal, from the or each said connected processor debug unit to said system debug unit via said signal path; wherein: at least one said debug information signal is common to both said first and said second processing units and is generated by each unit in dependence upon a value used in the processing unit concerned, said value used by said first processing unit having more bits than said value used by said second processing unit; and said processor debug unit corresponding to said second processing unit pads its said value such that said common debug information signal has the same number of bits when generated in either of said first and second processing units.

In the fourth and fifth aspects of the invention there may be one system debug unit provided in common for all the processor debug units or one system debug unit per processor debug unit. In the third, fourth and fifth aspects of the invention, the signal path

width may be sufficient that each message can be sent in a single cycle.

According to a sixth aspect of the present invention there is provided a method of debugging a program for a processor using processor circuitry embodying the first, third, fourth or fifth aspect of the present invention connected operatively to said further circuitry, said method comprising: (a) receiving in said further circuitry said one or more further signals output by said system debug unit; and (b) using the received signal(s) to debug said program.

According to a seventh aspect of the present invention there is provided the debugged program produced by a debugging method embodying the sixth aspect of the invention. The debugged program may be stored on a recording medium such as a read-only memory (ROM) so that it can be supplied and used with production devices not including the system debug unit.

Reference will now be made, by way of example, to the accompanying drawings in which:

Fig. 1, discussed hereinbefore, is a schematic view of parts of a conventional debugging system;

Fig. 2, also discussed hereinbefore, is a schematic view showing parts of the Fig. 1 debugging system contained within an SOC device;

Fig. 3 shows parts of processor circuitry embodying the present invention;

Fig. 4 presents a table for use in explaining various debug information signals generated in the Fig. 3 processing circuitry;

Fig. 5 is a diagram for use in explaining operation of message transfer circuitry in the Fig. 3 processor circuitry;

Fig. 6 shows parts of processor circuitry according to a first embodiment of the present invention; and

Fig. 7 shows parts of processor circuitry according to a second embodiment of the present invention.

Fig. 3 shows parts of processor circuitry 20 embodying the present invention. The processor circuitry 20 includes a processing unit (PU) 22. When the processor circuitry 20 is an SOC device, for example, the processing unit 22 may be a processor core. As is well known, such a processor core may itself contain more than one processing unit. In particular, the processor core may be a VLIW processor core in which the different processing units execute different instructions belonging to the same VLIW instruction packet in parallel with one another. Alternatively, or in addition, the processor core may have a so-called clustered architecture. In this kind of architecture small groups of processing units are clustered around separate partitions of a register file. The present applicant's Opus architecture is one example of such a clustered architecture.

The processor core 20 is provided with a processor debug unit (PDU) 24 which contains one or more debug event detectors (not shown) used to detect one or more preselected debug events occurring in the processor core 22. These debug events may be, for example, instruction breakpoints or watchpoints (IBWs), data breakpoints or watchpoints (DBWs), branch events requiring generation of branch trace messaging (BTM), data acquisition messaging events requiring generation of data acquisition messaging (DQM), or other CPU events (cache hit, cache miss, etc). The types of debug event detectors which can be provided in the PDU are not limited in this invention.

The constitution of each type of event detector is dependent on the event being detected, but is well-known in the art. For example, an IBW event detector

comprises address comparators which compare the current program counter value with a single programmable address or with programmable start and end addresses defining an address range. The comparators match when the current program counter value is equal to the single address or is within the specified address range. A DBW event detector is constituted in generally the same way as an IBW event detector except that in place of instruction addresses data addresses are compared by the comparators. The DBW comparators match when an executing instruction performs a load or store operation at a single specified address or within a specified range of addresses.

Fig. 4 shows examples of various debug information signals generated by the PDU 24 in response to different debug events. Fig. 4 also indicates, for each such signal, a typical number of bits making up the signal, and the intended purpose of the signal. The final five columns of Fig. 4 indicate various debug events which cause the different debug information signals to be generated.

The first debug information signal is a program counter (PC) value signal PC_VALUE. This signal specifies the PC value of the last retired instruction.

The second debug information signal is a current address number (CASN) signal CASN_VALUE. This signal specifies a value held in a register of the processing unit 22 which is used by the processing unit's virtual memory mechanism to select which translation lookaside buffer (TLB) entries are available to the executing instruction stream. CASN_VALUE indicates the CASN value held in the register concerned when the last instruction was retired.

The third debug information signal is a data value signal DATA_VALUE. If the last retired instruction triggered a DBW event, the DATA_VALUE signal specifies

the data value read or written by that instruction. The DATA_VALUE signal may be 8, 16, 32 or 64 bits according to the data value being accessed by the triggering load/store instruction.

The fourth debug information signal is a DBW address signal DBW_ADDRESS. If the last retired instruction triggered a DBW event, the DBW_ADDRESS signal contains the address at which the load/store operation was performed.

The fifth debug information signal is a IBW event identifier signal IBW_HITMASK. When (as is usually the case) the PDU has a plurality of IBW event detectors the IBW_HITMASK signal indicates which (if any) of the IBW event detectors has triggered. In this example, it is assumed that there are four IBW event detectors, so that the IBW_HITMASK signal has four bits corresponding respectively to the four IBW event detectors.

The sixth debug information signal is a DBW event identifier signal DBW_HITMASK. This is similar to the IBW_HITMASK signal except that it indicates which (if any) of the DBW event detectors has triggered.

The seventh debug information signal is a next PC value signal PCDEST_VALUE. When a branch instruction is executed, the PCDEST_VALUE signal indicates the address of the instruction to which the branch will be made.

The eighth debug information signal is a DQM value signal DQM_VALUE. If the last retired instruction was a write to a DQM register, the DQM_VALUE signal specifies the value written to that register. The DQM_VALUE signal has the same number of bits as the DQM register that has been written to, for example 32 bits.

As shown in Fig. 4, the debug information signals which need to be transferred from the PDU 24 to the SDU 26 vary for different debug events.

On normal instruction execution, none of the

signals needs to be transferred.

When an instruction or data breakpoint event occurs (IBW/DBW exception) the signals IBW_HITMASK and DBW_HITMASK are supplied by the PDU to the SDU.

When an IBW event occurs and an IBW trace message is to be generated by the SDU, the PDU supplies to the SDU the PC_VALUE signal, the CASN_VALUE signal and the IBW_HITMASK and DBW_HITMASK signals.

When a DBW event occurs and a DBW trace message is to be generated by the SDU, the PDU supplies to the SDU the PC_VALUE, CASN_VALUE, DATA_VALUE, IBW_HITMASK and DBW_HITMASK signals.

When a DQM event occurs, i.e. the processing unit 22 writes a DQM data value to one of the DQM registers in the PDU, the PDU supplies to the SDU the PC_VALUE, CASN_VALUE and DQM_VALUE signals.

When a BTM event occurs so that the SDU is required to generate a BTM, the PDU supplies to the SDU the PC_VALUE, CASN_VALUE and PCDEST_VALUE signals.

The way in which the debug information signals set out in Fig. 4 are transferred from the PDU 24 to the SDU 26 will now be explained. As shown in Fig. 3, the PDU 24 comprises a first communication unit 25 and the SDU 26 comprises second and third communication units 28 and 29 and a message buffer 30. The PDU 24 is connected to the SDU 26 by interconnection circuitry 32 which provides first, second and third signal paths 34, 36 and 38 therebetween. The communication units 25, 28 and 29, message buffer 30 and interconnection circuitry 32 together form message transfer circuitry.

The first signal path 34 is used to transfer a message data signal INFOVALUE from the PDU to the SDU. The second signal path 36 is used to transfer a valid signal INFOVALID from the PDU to the SDU. The third signal path 38 is used to transfer control signals (denoted collectively in Fig. 3 by CONTROL) from the

SDU to the PDU. In one embodiment of the present invention these control signals include a INFOGRANT signal, an ENTER_DEBUG_MODE signal and a STALL signal (see Table 1 below).

The PDU also has a clock path (not shown) connecting it to the SDU 26. This clock path carries a clock signal PDU_CLK.

Table 1 below indicates the directions, widths, numbers of bits and intended purposes of the signals transferred by the interconnection circuitry 32. These signals are used by the message transfer circuitry to form a bus which carries multi-bit information in the form of messages from the PDU 24 to the SDU 26.

Table 1

NAME	DIRECTION	WIDTH	PURPOSE
PDU_CLK	PDU to SDU	1	Rising edge indicates that other PDU to SDU signals are now valid.
INFOVALID	PDU to SDU	2	Indicates status of INFOVALUE signals.
INFOVALUE	PDU to SDU	scalable from 1..N	Carries multi-bit data from PDU to SDU.
INFOGRANT	SDU to PDU	1	When inactive (0): SDU has not granted access to the interconnection circuitry 32. Thus the PDU must not start a message transfer or drive the INFOVALID/INFOVALUE signals high. When active (1): SDU is granting access to the interconnection circuitry 32. Thus the PDU is free to initiate a message transfer and to drive the INFOVALID/INFOVALUE signals high or low as required to perform a message transfer.
ENTER_DEBUG_MODE	SDU to PDU	1	When high, requests that the PDU enters a special Debug Mode in which the processing unit 22 executes debugger software.
STALL	SDU to PDU	1	When high, requests that the PDU stalls the processing unit 22 as the SDU cannot cope with the v lume of trace being generated.

The INFOVALUE signal is used to carry all data from the PDU to the SDU. Although in principle the INFOVALUE signal may have any number of bits down to just one bit, in practice a preferred number of bits in the INFOVALUE signal is between 8 and 32 bits, representing a reasonable tradeoff between numbers of wires required and available data transfer bandwidth. In this example, it will be assumed that the INFOVALUE signal has 32 bits.

The INFOVALID signal indicates the state of data being transferred from the PDU to the SDU via the first signal path 34. This signal is a two-bit signal which can have one of four values corresponding respectively to four different message transfer states which can exist in the message transfer circuitry, namely an invalid state, a start state, a mid state and an end state.

When the INFOVALID signal has the value "00" this denotes the invalid state. In this state there is no valid data being transferred from the PDU to the SDU, i.e. the INFOVALUE signal is ignored by the SDU. When the INFOVALID signal has the value "01" this denotes the start state signifying the start of transfer of a new message. When the INFOVALID signal has the value "10" this denotes the mid state signifying an intermediate stage of a message transfer. When the INFOVALID signal has the value "11" this denotes the end state, signifying the end of transfer of a message.

Because at least some of the debug events described above (in particular events leading to trace message generation) call for several of the debug information signals shown in Fig. 4 to be sent from the PDU to the SDU, and in total (or even individually) those signals may have more bits than the first signal path 34 has lines, a message transferred from the PDU

to the SDU may require more than one cycle to be transferred.

The message transfer circuitry is a finite state machine which undergoes a series of state changes as a message transfer progresses. The value of the INFOVALID signal reflects the current state of the message transfer circuitry and is controlled by the first communication unit 25 in the PDU 24 in dependence upon the control signals received from the SDU 26, in particular the INFOGRANT signal, as will be described later. The second and third communication units 28 and 29 can refer to the INFOVALID signal to check the message transfer state, and in particular to distinguish between different cycles of a message.

The valid changes of state are as follows:

(a) Invalid to start: this denotes the start of a first cycle of a message. A message may have only a single cycle, in which case this state change denotes the start of the only cycle of the message. This state change is only permissible when the INFOGRANT signal supplied by the SDU to the PDU is active (1).

(b) Start to mid: in a message having three or more cycles this denotes the beginning of the second cycle. This state change is not used to denote the start of the second cycle in a message having only two cycles (see below (c)).

(c) Start to end: in the special case of a message having only two cycles, this denotes the beginning of the final cycle (second cycle).

(d) Start to invalid: in the case of a single-cycle message, this denotes the end of the message.

(e) Mid to mid: in a message having four or more cycles this denotes the beginning of any cycle other than the first, second and last cycles.

(f) Mid to end: in a message having three or more cycles this denotes the beginning of the last cycle.

(g) End to invalid: in the case of any message other than a single-cycle message, this denotes the end of the message.

A message must always end with the invalid state so that between two successive messages the invalid state must be entered.

Fig. 5 is a diagram for illustrating operation of the Fig. 3 message transfer circuitry.

Initially, in a step S1 the circuitry is in an idle state, i.e. no debug exception has been raised and no IBW/DBW/BTM/DQM event requiring a trace message has occurred. This corresponds to the invalid state described above.

If, in step S1, an IBW/DBW debug exception is raised the current state of the circuitry is checked. If the state is the invalid state and the INFOGRANT signal is active a message transfer is possible and the processing continues with a step S2 in which a message including the IBW_HITMASK and DBW_HITMASK signals is transmitted via the first signal path 34 to the SDU 26.

In step S2 the current state is changed to the start state and the INFOVALID signal becomes "01". Thus, the second communication unit 28 in the SDU knows that there are valid signals on the first signal path 34. These signals are received and stored by the unit 28 SDU in the message buffer 30.

In the case of an IBW/DBW exception, there are no further signals to transmit to the SDU, and so processing returns to S1, i.e. the state is changed back from the start state to the invalid state. The SDU sees the resulting change in the INFOVALID signal and knows that the message transfer is complete.

If in step S1 an IBW/DBW/BTM/DQM debug event is detected which requires a trace message to be generated, the current state of the circuitry is checked as for a debug exception. If a message

transfer can proceed, processing moves to step S3 in which a first cycle of the message transfer is carried out. In this first cycle the PC_VALUE signal is transmitted from the PDU to the SDU. The current state is changed from the invalid state to the start state, and the second communication unit 28 knows from the change in the INFOVALID signal to "01" that there are valid signals present on the first signal path 34. It can be seen that from Fig. 4 that transmission of the PC_VALUE signal is required for all events other than the debug exception event, and accordingly step S3 is carried out for each of these other events.

If the event is an IBW event requiring trace message generation, step S4 is carried out after step S3. In this step, the CASN_VALUE and IBW_HITMASK signals are transmitted to the SDU in a second cycle of the message transfer. The current state is therefore changed from start to end, and the INFOVALID signal is changed to "11". Thus, the SDU knows that this is the final cycle of the current message. Processing then returns to step S1.

If in step S3 the event requiring trace message generation is a DBW, DQM or BTM event, processing proceeds to step S5 for the second cycle of the message transfer. In step S5, similarly to the step S4, the CASN_VALUE signal is transmitted to the SDU. In step S5, however, both the IBW_HITMASK and DBW_HITMASK signals are also sent in parallel with the CASN_VALUE signal. This is possible because the width of the first signal path 34 is greater than the total of the number of bits (8+4+4) in the three signals CASN_VALUE, IBW_HITMASK and DBW_HITMASK.

As explained later, the IBW_HITMASK signal is included in the signals transmitted in step S5 because it is possible that an IBW event requiring trace message generation and another event (e.g. a DBW event)

requiring trace message generation may occur in the same PDU clock cycle. In this case, because step S5 transmits all the information transmitted by step S4, step S4 can be omitted, saving one cycle of the overall message transfer process.

The transmission in step S5 is not the final cycle of the message, and accordingly the current state of the circuitry is changed from the start state to the mid state, and the INFOVALID signal is changed to "10".

After step S5 processing may proceed to any one of steps S6, S7 and S9 depending on the debug event requiring trace message generation.

Step S6 is carried out when the debug event is a BTM event. In this case, the final cycle of the message transmits the PCDEST_VALUE signal to the SDU. The current state is changed from the mid state to the end state, and the INFOVALID signal is changed to "11". Processing then returns to step S1.

When the debug event requiring trace messaging generation is a DBW event, processing proceeds to step S7 in which the DATA_VALUE signal is transmitted to the SDU in a third cycle of the message. In the case of a DBW event this is still not the last cycle of the message, and so the mid state is maintained (INFOVALID signal maintained at "10"). Processing proceeds to step S8 in which the DBW_ADDRESS signal is transmitted to the SDU. This is the last cycle of the message and so the current state is changed from mid to end, and the INFOVALID signal is changed to "11". Processing then returns to step S1.

Finally, when the debug event requiring trace message generation is a DQM event, processing proceeds to step S9 in which the DQM_VALUE signal is transmitted to the SDU. This is the last cycle of a DQM message and, accordingly, the current state is changed from mid to end and the INFOVALID signal is changed to "11".

Processing then returns to step S1.

In Fig. 3 the message buffer 30 has a capacity equal to the size of the largest message which can be received from the PDU 24. This largest message size is equal to the product of the width of the first signal path 34 and the maximum number of message transfer cycles in the largest message. In Fig. 5 there are four cycles in a DBW message, as shown in steps S3, S5, S7 and S8. Thus, in this case the message buffer needs to have a capacity of at least 128 bits.

The third communication unit 29 determines that a complete message is available in the message buffer 30 when the message transfer circuitry changes from the end state to the invalid state or from the start state to the invalid state. At this point the third communication unit deactivates the INFOGRANT signal so that the first communication unit 25 will not send any further messages. The third communication unit 29 then empties the message buffer 30, generates an external message which it outputs to the debug link 7 and then reactivates the INFOGRANT signal.

If in step S1 the INFOGRANT signal is inactive (0) the third communication unit 29 is currently emptying the message buffer 30 and therefore cannot receive another message for the time being. In this condition, the PDU 24 preferably stalls the processing unit 22 in the event that the INFOGRANT signal is found to be inactive when it is desired to initiate a message transfer.

In one embodiment, the PDU 24 contains a response control register which controls the PDU's response in step S1 to an inactive INFOGRANT signal. In dependence upon the value held in the response control register the PDU 24 either stalls the processing unit 22 until the INFOVALID signal changes to "00" (the invalid state) or abandons generation of the new message. The

response control register value may be transmitted together with the IBW_HITMASK and DBW_HITMASK signals to the SDU in steps S2, S4 and S5 so that the external debugger software is aware of whether the processing unit 22 has been stalled or information has been lost. In this case, the register value is preferably cleared automatically to 0 whenever the IBW_HITMASK and/or DBW_HITMASK signals are transmitted.

Some events requiring generation of trace messages, for example a DQM event, may be arranged to always result in stalling of the processing unit whenever the INFOVALID signal has the invalid state, irrespective of how the response control register has been set.

As described above, the message transfer circuitry in Fig. 3 generates messages including the debug information signals and employs a single signal path (the first signal path 34) of the interconnection circuitry 32 to transfer different messages, at different respective times, from the PDU to the SDU. Conventionally, each different debug information signal has been provided with its own signal path, separate from that of each other debug information signal. This has meant either that the number of different debug information signals supplied to the SDU had to be limited to keep the overall scale of the interconnection circuitry down or that large-scale interconnection circuitry had to be provided, limiting the space available for other circuitry. Processor circuitry embodying the invention avoids both these limitations and enables a rich set of debug information signals to be transferred form the PDU to the SDU without large-scale interconnection circuitry.

The messages transferred by the message transfer circuitry can be of different lengths and can include any one or more of the debug information signals. Some

or all messages can be greater in size than a width of the single signal path.

The messages transferred from the PDU to the SDU shown in Fig. 5 are optimised taking into account matters such as which (if any) debug events can occur in the same PDU clock cycle (processor cycle) and the priorities to be accorded to those different events. In particular:

(1) When a debug exception event occurs (instruction breakpoint or data breakpoint) the instruction which triggered the exception is not committed and thus no IBW, DBW, BTM or DQM trace messages can be generated from the same event. Thus, the only information required is provided by the IBW_HITMASK and DBW_HITMASK signals.

(2) As explained above, if both IBW and DBW trace messages should be generated for events occurring in the same clock cycle, it is acceptable to generate only a DBW trace message, as this contains a superset of the debug information signals (PC_VALUE, CASN_VALUE and IBW_HITMASK) required for a IBW trace message.

(3) Similarly, if both DQM and IBW trace messages are required for events occurring in the same clock cycle it is acceptable to generate only a DQM trace message as this contains a superset of the debug information signals required by the IBW trace message.

(4) If both DQM and DBW messages are required for events occurring in the same clock cycle it is acceptable to generate only a DQM trace message as this contains all the necessary debug information signals (PC_VALUE, CASN_VALUE and DBW-HITMASK). The DBW_ADDRESS signal is not required because it is implicit from the DQM event that a DQM register was written to. Similarly, the DATA_VALUE signal is not required as the DQM_VALUE signal must provide the value written to the DQM register.

(5) For the same reasons as in (4) above, when DQM, IBW and DBW messages are required for events occurring in the same clock cycle, it is acceptable to generate only a DQM trace message.

(6) As with cases (2) and (3) above, if trace messages are required for BTM and IBW events occurring in the same clock cycle it is acceptable to generate only a BTM trace message as this contains a superset of the debug information signals (PC_VALUE, CASN_VALUE and IBW_HITMASK) required by the IBW trace message.

(7) In a processing unit having only a single instruction issue slot a BTM trace message and a DBW trace message cannot both be generated in the same clock cycle as a single instruction cannot branch and modify memory. In a processing unit having plural instruction issue slots (for example a VLIW processing unit) it is possible that BTM and DBW events requiring trace generation could occur in the same clock cycle. In this case, the PDU 24 needs to be provided with buffering sufficient to hold the maximum number of messages which can be generated from events occurring in the same clock cycle, so that the buffered messages can then be transferred one by one to the SDU. For example, referring to Fig. 5, the BTM message could be transmitted first by carrying out steps S1, S3, S5 and S6. Then a DBW trace message could be generated by carrying out the steps S1, S3, S5, S7 and S8.

(8) Similarly, in a single-issue processing unit, trace messages cannot be generated for BTM and DQM events occurring in the same clock cycle. However, in a processing unit having plural issue slots, BTM and DQM events requiring trace messages can occur in the same clock cycle. As explained at (7) above, it is then necessary to provide sufficient buffering in the PDU 24 to cope with all of the messages that can be generated for events occurring in the same cycle.

The number of steps in Fig. 5 is minimised taking account of the properties (1) to (8) set out above. Similarly, from a consideration of these properties it is evident that an optimal number of message transfer cycles can be obtained by setting the width of the first signal path 34 so that it is sufficient to carry in a single message cycle either: (a) any one of the PC_VALUE, PCDEST_VALUE, DATA_VALUE or DBW_ADDRESS signals by itself; or (b) all of the CASN_VALUE, IBW_HITMASK and DBW_HITMASK signals together.

In the Fig. 4 example, it follows from (a) that the first signal path 34 should have at least 32 bits. 32 bits are also sufficient to deal simultaneously with the signals mentioned at (b).

It will be appreciated by those skilled in the art that if the width of the first signal path 34 is greater than that needed to cater for the signals in (a) or (b) Fig. 5 could be modified to have less steps, i.e. more signals transmitted simultaneously. This could result in some or all of the messages having fewer cycles than in Fig. 5.

Alternatively, if the first signal path 34 is too narrow to cope with the signals at (a) and/or (b) in one message cycle, Fig. 5 will be modified to include more steps, with some or all of the steps involving transmission of fewer signals than in Fig. 5.

In Fig. 3 the SDU 26 has a single message buffer 30. However, it would be possible to provide a series of message buffers 30 in a pipeline configuration between the second and third communication units 28 and 29. In this case, the message transfer circuitry could permit a new message to be sent whenever a free message buffer is available. Thus, the SDU 26 could receive a message from the PDU 24 at the same time as transmitting an already-received message to the debug link 7.

As mentioned in the introduction, processor circuitry may include more than one processing unit, so that debug information may need to be supplied from each of the plurality of processing units to the debug link 7. As will be described in more detail later in the present specification, the processing units included within the processor circuitry may not all be of the same type. For example, some processing units may use "narrower" data than others, for example only 16-bit program counter values instead of 32-bit program counter values. Similarly, some processing units may not use or make available certain types of debug information signal. For example a processing unit which does not support virtual memory will not have the CASN_VALUE debug information signal to supply as one of its debug information signals.

Embodiments of the present invention can enable two or more processing units of the same or different types to supply debug information signals efficiently to a shared or common SDU within the processor circuitry, the SDU then generating external messages for supply to the debug link 7.

A first such embodiment is shown in Fig. 6. In Fig. 6, processor circuitry 120 comprises first and second processing units 22_0 and 22_1. Each processing unit is provided with a PDU 24 as in Fig. 3. That PDU in turn comprises a first communication unit 25 as in Fig. 3.

The processor circuitry 120 further comprises an SDU 126 which comprises second communication units 128_0 and 128_1, message buffers 130_0 and 130_1, and a single third communication unit 129. The third communication unit 129 is connected to a debug link 7.

In Fig. 6 the first communication unit 25_0 of the PDU 24_0 is connected by first interconnection circuitry 32_0 to the second communication unit 128_0. The first

and second communication units 25_0 and 128_0 and the first interconnection circuitry 32_0 together constitute first message transfer circuitry provided for the first processing unit 22_0. The first interconnection circuitry 32_0 comprises first, second and third signal paths 34_0, 36_0 and 38_0 corresponding respectively to the paths 34, 36 and 38 in Fig. 3.

The first communication unit 25_1 and the second communication unit 128_1 are connected via second interconnection circuitry 32_1. The first communication unit 25_1, second communication unit 128_1 and the second interconnection circuitry 32_1 together form second message transfer circuitry provided for the second processing unit 22_1. The third communication unit 129 is common to both the first and second message transfer circuitry.

The first and second message transfer circuitry each operate in the same way as the message transfer circuitry in Fig. 3. The third communication unit 129 operates in basically the same way as the unit 29 in Fig. 3 but in Fig. 6 it is connected to both of the message buffers 130_0 and 130_1. Thus the unit 129 monitors the states of all of the message buffers and generates messages for supply to the debug link 7 when any of the message buffers contains a complete message.

Each message buffer 130 maintains a valid message signal VM which is initially deactivated. When the first or second message transfer circuitry completes transfer of a message, i.e. when the state of the transfer circuitry changes from the end state to the invalid state or from the start state to the invalid state, the transfer circuitry concerned activates the VM signal in the corresponding message buffer 130. The third communication unit 129 monitors the respective VM signals of the different message buffers 130 and, when any one of them has an active VM signal, the third

communication unit 129 deactivates the INFOGRANT signal for that message transfer circuitry and empties its message buffer. An external message is sent to the debug link 7 reflecting the contents of the emptied message buffer. The third communication unit then supplies a reset signal RST to the emptied message buffer which results in the VM signal of that buffer being deactivated. The third communication unit also then reactivates the INFOGRANT signal for the message transfer circuitry concerned.

In the Fig. 6 embodiment each PDU is provided with its own independent message transfer circuitry having independent first, second and third signal paths for transferring signals between itself and the SDU. It is also possible, however, to have the PDUs share parts of the interconnection circuitry, for example parts of the first and second signal paths.

Fig. 7 shows processor circuitry 220 which includes first and second processing units 22_0 and 22_1, each having its own PDU 24_0 or 24_1. Each PDU includes a first communication unit 225.

The processor circuitry 220 also includes a SDU 226. The SDU contains a single second communication unit 220, a message buffer 230 and a single third communication unit 229. Each first communication unit 225 is connected via a first signal path portion 234 to first combiner circuitry 240. Similarly, each first communication unit 225 is connected via a second signal path portion 236 to second combiner circuitry 242. The first combiner circuitry 240 has an output connected via a first bus 244 to an input of the second communication unit 228 in the SDU 226. Similarly the second combiner circuitry 242 has an output connected via a second bus 246 to an input of the second communication unit 228.

The first combiner circuitry 240 comprises, for

example, m n-bit OR gates, where m is the width of the first bus and n is the number of processing units 22 to be connected to the SDU 226. m is, for example, 32. Each of the first and second signal path portions 234 is also of width m, and each of the m OR gates has its n inputs connected to corresponding lines within the n different first signal path portions 234. Each OR gate has its output connected to one line of the first bus 244.

The second combiner circuitry 242 comprises, for example, a single n-bit OR gate. The n inputs of the OR gate are connected respectively to the n different second path portions 236. The output of the OR gate is connected to the second bus 246.

Each first communication unit 225 also has a third signal path 238 connecting it to the second communication unit 228. In this embodiment the third signal path is a bi-directional signal path and, as well as enabling the INFOGRANT, ENTER_DEBUG_MODE and STALL signals shown in Table 1 above to be transmitted from the SDU to the PDU, enables the PDU to send an access request signal INFOREQUEST to the SDU.

The INFOREQUEST signal is activated by a first communication unit 225 when it wishes to have access to the first and second buses 244 and 246. The second communication unit 228 monitors all of the third signal paths for activation of an INFOREQUEST signal from one of the first communication units. Upon detecting that one of the first communication units 225 has activated its INFOREQUEST signal the second communication unit 228 checks whether the first and second buses are already in use for transfer of a message from another first communication unit. The third communication unit 229 also checks whether the message buffer 230 is empty. If the first and second buses are not already in use, and the message buffer 230 is empty, the third

communication unit 229 activates the INFOGRANT signal to the requesting first communication unit.

If the third communication unit 229 denies the requesting first communication unit access to the first and second buses, the invalid state is maintained for that first communication unit, and accordingly its INFOVALID signal remains inactive. In response to this, as described earlier, the PDU stalls its processing unit or discards the message.

The Fig. 7 embodiment has the advantage that its interconnection circuitry 232 is reduced in scale compared to that of Fig. 6. However, because the different PDUs must share the interconnection circuitry 232, it may be necessary to stall the processing units more often and/or to discard trace messages more often in the Fig. 7 embodiment than in the Fig. 6 embodiment. It is possible to alleviate this problem in the Fig. 7 embodiment by replacing the single message buffer 230 by a series of pipelined message buffers as described earlier with reference to Fig. 3.

In the Figs. 6 and 7 embodiments the processing units do not have to be of the same type, and the PDUs can be different from one another, for example they may have different numbers and or types of debug event detector. Each PDU must, of course, include a first communication unit capable of carrying out message transfer with the SDU, and in this sense the PDUs can be said to be mutually-similar or "heterogeneous". When the processor circuitry has processing units of different types, the message transfer circuitry must be designed to cope with the signals generated by the PDU of each different processing unit. This can be achieved by defining standard signal formats for all the PDUs to use for their debug information signals.

In particular, it is necessary to choose standard widths for certain signals. For example, if one

processing unit uses 16-bit PC values and another uses 32-bit PC values, the standard width of the PC_VALUE signal may be set at 32-bits, and the PDU for the processing unit using 16-bit PC values must still generate a PC_VALUE signal having 32 bits, for example by padding the upper 16 bits.

Similarly, if one of the processing units supports virtual memory but another one does not, the standard debug information signals sent in steps S4 and S5 in Fig. 4 may still include the CASN_VALUE. In this case, the PDU of the processing unit which does not support virtual memory must still generate a "dummy" CASN_VALUE signal for supply to the first communication unit so that the overall standard message structure is obeyed.

As described above, in processor circuitry embodying the present invention the SDU is separated by interconnection circuitry from the or each PDU. This allows the SDU to be omitted in a production device if this is desirable for security or space reasons.

The extent of the interconnection circuitry is desirably small, so it has little impact on the area or speed of the processor circuitry.

In addition, it is possible to support two or more processing units by providing the different processing units with heterogeneous PDUs connected to a single SDU. Apart from involving less circuitry on-chip, it is also possible in this way to provide a common off-chip interface and feature set.

Finally, debugger software for processor circuitry embodying the present invention is simple to program and to present as a "user model" because the processing unit(s) have the same trace messages generated externally.

CLAIMS:

1. Processor circuitry comprising:

a processing unit;

a processor debug unit provided for said processing unit and operable to generate one or more debug information signals for said processing unit;

a system debug unit, connected with said processor debug unit by a signal path, and operable to receive each said debug information signal and to output one or more further signals derived therefrom to further circuitry; and

message transfer means operable to transfer one or more messages, each including at least one said debug information signal, from said processor debug unit to said system debug unit via said signal path, at least one said message having a number of bits greater than a width of said signal path, and said message transfer means being operable to transfer that message from said processor debug unit to said system debug unit in a series of message cycles.

2. Processor circuitry as claimed in claim 1, wherein every one of said debug information signals generated by said processor debug unit is included in such a message, and said message transfer means are operable to employ said signal path to transfer all the messages from the processor debug unit to the system debug unit.

3. Processor circuitry as claimed in claim 1 or 2, wherein one said message has a different number of said message cycles from another said message.

4. Processor circuitry as claimed in claim 1, 2 or 3, wherein the said message transfer means are operable to transfer each such message only when the said debug information signal to be included in that message is generated by the said processor debug unit.

5. Processor circuitry as claimed in any

preceding claim, wherein at least one said message includes two or more of said debug information signals.

6. Processor circuitry as claimed in any preceding claim, wherein:

said message transfer means are operable to transfer a first-debug-event message, including one or more of said debug information signals, in response to a first debug event occurring in the said processing unit and to transfer a second-debug-event message, including one or more of said debug information signals, in response to a second debug event occurring in said processing unit; and

said first-debug-event message includes each of said one or more debug information signals included in said second-debug-event message, whereby, when said first and second debug events occur simultaneously in said processing unit, said second-debug-event message is not transferred from said processor debug unit to said system debug unit.

7. Processor circuitry as claimed in claim 6, wherein said first-debug-event message includes at least one said debug information signal that is generated in response to said second debug event but not generated in response to said first debug event.

8. Processor circuitry as claimed in any preceding claim, wherein said message transfer means are operable to carry out a limited set of transfer operations to transfer said messages, each said transfer operation serving to transfer at least one said debug information signal from said processor debug unit to said system debug unit, and at least two said messages involving the same one of said transfer operations.

9. Processor circuitry as claimed in claim 8, wherein each said transfer operation occupies one said message cycle.

10. Processor circuitry as claimed in any preceding claim, wherein said message transfer means are operable to check, prior to initiating transfer of one of said messages, whether said system debug unit is ready to receive said message and to permit the transfer to proceed only when the system debug unit is ready.

11. Processor circuitry as claimed in claim 10, wherein the message transfer means cause a grant signal to be sent from the system debug unit to the processor debug unit to indicate that the system debug unit is ready to receive said message.

12. Processor circuitry as claimed in claim 10 or 11, wherein, if said system debug unit is not ready to receive the message, said processor debug unit is operable to cause said processing unit to be stalled until the system debug unit is ready to receive the message.

13. Processor circuitry as claimed in any one of claims 10 to 12, wherein, if the system debug unit is not ready to receive the message, said processor debug unit is operable to abort the transfer of the message.

14. Processor circuitry as claimed in any one of claims 10 to 13, wherein, if the system debug unit is not ready to receive the message, the processor debug unit is operable selectively to cause its said processing unit to be stalled or to abort the message transfer.

15. Processor circuitry as claimed in claim 14, wherein the message transfer means are operable to include in one of said messages sent from the processor debug unit to the system debug unit an indication of which one of said stalling and said aborting was selected.

16. Processor circuitry as claimed in any preceding claim, wherein said message transfer means

are switchable between at least an idle state, in which no message is being transferred from said processor debug unit to said system debug unit, and one or more working states in which transfer of a message is carried out.

17. Processor circuitry as claimed in claim 16, wherein one said working state is a start state, and said message transfer means are switched from said idle state to said start state when a message transfer is started.

18. Processor circuitry as claimed in claim 16 or 17, wherein one of said working states is an end state, and said message transfer means are switched from said end state to said idle state when a message transfer is finished.

19. Processor circuitry as claimed in claim 18 when read as appended to claim 17, wherein one of said working states is a mid state and, as a message transfer progresses, the message transfer means are switchable from said idle state to said start state to said mid state to said end state and back to said idle state.

20. Processor circuitry as claimed in any one of claims 16 to 19 when read as appended to claim 11, wherein said message transfer means are operable to commence transfer of a message only when said grant signal is received in said idle state.

21. Processor circuitry as claimed in any preceding claim, further comprising:

one or more further processing units; and

one or more further processor debug units, each provided for a corresponding one of said further processing units, and each operable to generate at least one debug information signal for its corresponding further processing unit;

wherein each said further processor debug unit is

also connected with said system debug unit by a signal path, and said message transfer means being operable, for each said further processor debug unit, to transfer one or more messages, each including at least one debug information signal generated by that processor debug unit, to the system debug unit via said signal path.

22. Processor circuitry as claimed in claim 21, wherein each said further processor debug unit has its own signal path to the system debug unit, independent of the signal path of each other processor debug unit, and said signal path of each said further processor debug unit is employed to transfer such messages from that processor debug unit alone to the system debug unit.

23. Processor circuitry as claimed in claim 21, having a shared signal path which is employed by the message transfer means to transfer such messages from all of said processor debug units to said system debug unit.

24. Processor circuitry as claimed in any preceding claim, wherein:

said message transfer means comprise a first communication unit in the or each said processor debug unit for transmitting each message from that processor debug unit, and at least one second communication unit in said system debug unit for receiving each transmitted message; and

said system debug unit further comprises a message buffer in which each received message is stored.

25. Processor circuitry as claimed in claim 24 when read as appended to claim 22, wherein said message transfer means comprise one said second communication unit and one said message buffer for each processor debug unit included in the processor circuitry.

26. Processor circuitry as claimed in claim 24 when read as appended to claim 23, wherein:

a single said second communication unit is provided in said system debug unit; and

each said first communication unit is operable to send a request signal to the system debug unit when it wishes to send a message thereto, and said system debug unit is operable to supply a refusal signal to the requesting first communication unit if said shared signal path is already in use by another one of said first communication units.

27. Processor circuitry as claimed in any one of claims 24 to 26, wherein the system debug unit further comprises a third communication unit for accessing a message stored in the or each said message buffer and for outputting said further signal to said further circuitry in dependence on said accessed message.

28. Processor circuitry as claimed in claim 27 when read as appended to claim 11, wherein said third communication unit is operable to generate said grant signal in dependence upon a state of the or each said message buffer.

29. Processor circuitry as claimed in any one of claims 24 to 28, wherein the or each said message buffer is capable of storing a plurality of received messages.

30. Processor circuitry as claimed in any one of claims 21 to 23, wherein a first one of said processing units is different from a second one of said processing units.

31. Processor circuitry as claimed in claim 30, wherein:

at least one said debug information signal available in said first processing unit is not available in said second processing unit; and

said processor debug unit corresponding to said second processing unit produces a dummy information signal which is included in place of said unavailable

debug information signal in one of said messages sent by that processor debug unit to said system debug unit.

32. Processor circuitry as claimed in claim 30 or 31, wherein:

at least one said debug information signal is common to both said first and said second processing units and is generated by each unit in dependence upon a value used in the processing unit concerned, said value used by said first processing unit having more bits than said value used by said second processing unit; and

said processor debug unit corresponding to said second processing unit pads its said value such that said common debug information signal has the same number of bits when generated in either of said first and second processing units.

33. Processor circuitry comprising:

a processing unit;

a processor debug unit provided for said processing unit and operable to generate one or more debug information signals for said processing unit;

wherein said processor debug unit comprises a communication unit having a signal path output adapted for connection to a signal path having a preselected width, and operable to output from said signal path output one or more messages, each including at least one said debug information signal, at least one said message having a number of bits greater than said preselected width, and said communication unit being operable to output that message from said signal path output in a series of message cycles.

34. Processor circuitry comprising:

a plurality of processing units;

a plurality of processor debug units each provided for a corresponding one of said processing units and operable to generate one or more debug information

signals for its said corresponding processing unit;

a system debug unit, connected with each of said processor debug units by interconnection means providing a signal path from each processor debug unit to the system debug unit, and operable to receive each said debug information signal generated by the plurality of the processor debug units and to output one or more further signals derived therefrom to further circuitry; and

message transfer means operable to transfer one or more messages, each including at least one said debug information signal, from each said processor debug unit to said system debug unit via that processor debug unit's said signal path to the system debug unit, messages from different processor debug units being transferred to the system debug unit at different respective times.

35. Processor circuitry comprising:

a plurality of processing units;

a plurality of processor debug units each provided for a corresponding one of said processing units and operable to generate one or more debug information signals for its said corresponding processing unit;

a system debug unit, connected with at least one said processor debug unit by a signal path, and operable to receive each said debug information signal generated by the or each connected processor debug unit and to output one or more further signals derived therefrom to further circuitry; and

message transfer means operable to transfer one or more messages, each including at least one said debug information signal, from the or each said connected processor debug unit to said system debug unit via said signal path;

wherein:

at least one said debug information signal

available in said first processing unit is not available in said second processing unit; and

said processor debug unit corresponding to said second processing unit produces a dummy information signal which is included in place of said unavailable debug information signal in one of said messages sent by that processor debug unit to said system debug unit.

36. Processor circuitry comprising:

at least first and second processing units;

a plurality of processor debug units each provided for a corresponding one of said processing units and operable to generate one or more debug information signals for its said corresponding processing unit;

a system debug unit, connected with at least one said processor debug unit by a signal path, and operable to receive each said debug information signal generated by the or each connected processor debug unit and to output one or more further signals derived therefrom to further circuitry; and

message transfer means operable to transfer one or more messages, each including at least one said debug information signal, from the or each said connected processor debug unit to said system debug unit via said signal path;

wherein:

at least one said debug information signal is common to both said first and said second processing units and is generated by each unit in dependence upon a value used in the processing unit concerned, said value used by said first processing unit having more bits than said value used by said second processing unit; and

said processor debug unit corresponding to said second processing unit pads its said value such that said common debug information signal has the same number of bits when generated in either of said first

and second processing units.

37. Processor circuitry as claimed in any one of claims 1 to 32, 34, 35 or 36, wherein the or each said processor debug unit, the or each said signal path, and said system debug unit are provided together on a chip.

38. A method of debugging a program for a processor using processor circuitry as claimed in any preceding claim connected operatively to said further circuitry, comprising:

(a) receiving in said further circuitry said one or more further signals output by said system debug unit; and

(b) using the received signal(s) to debug said program.

39. A method as claimed in claim 38, further comprising:

(c) modifying the program taking account of the received signal(s).

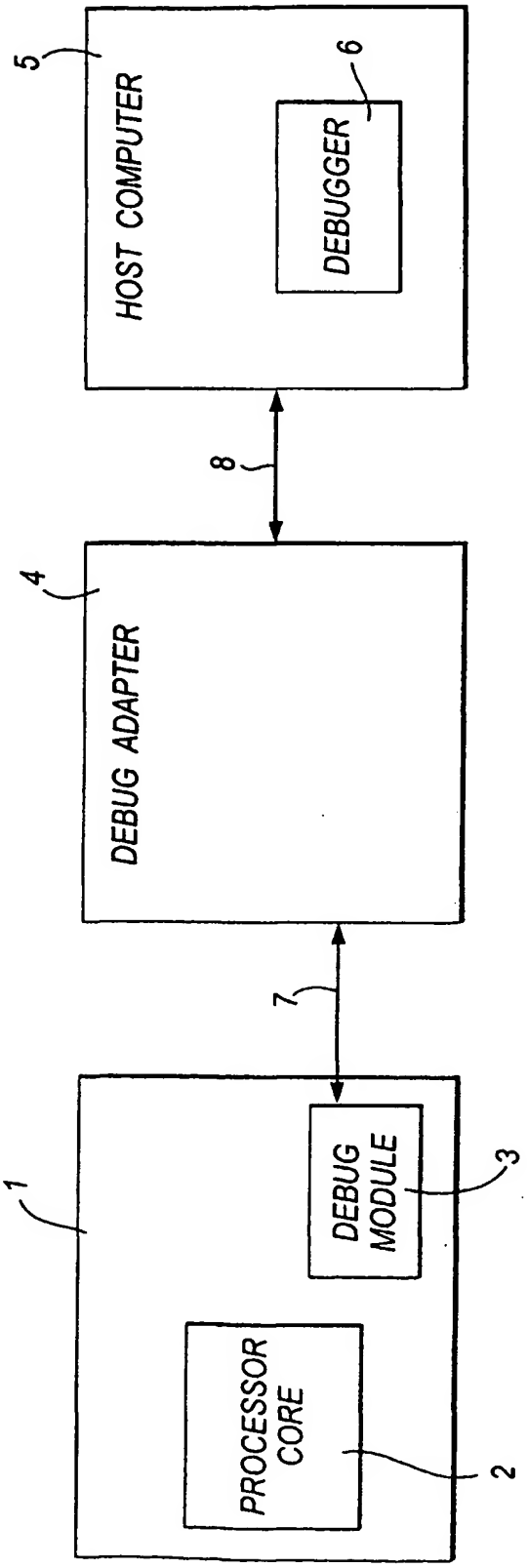

HOST COMPUTER
DEBUGGER
DEBUG ADAPTER
DEBUG MODULE
PROCESSOR CORE
1
2
3
4
5
6
7
8

Fig.1

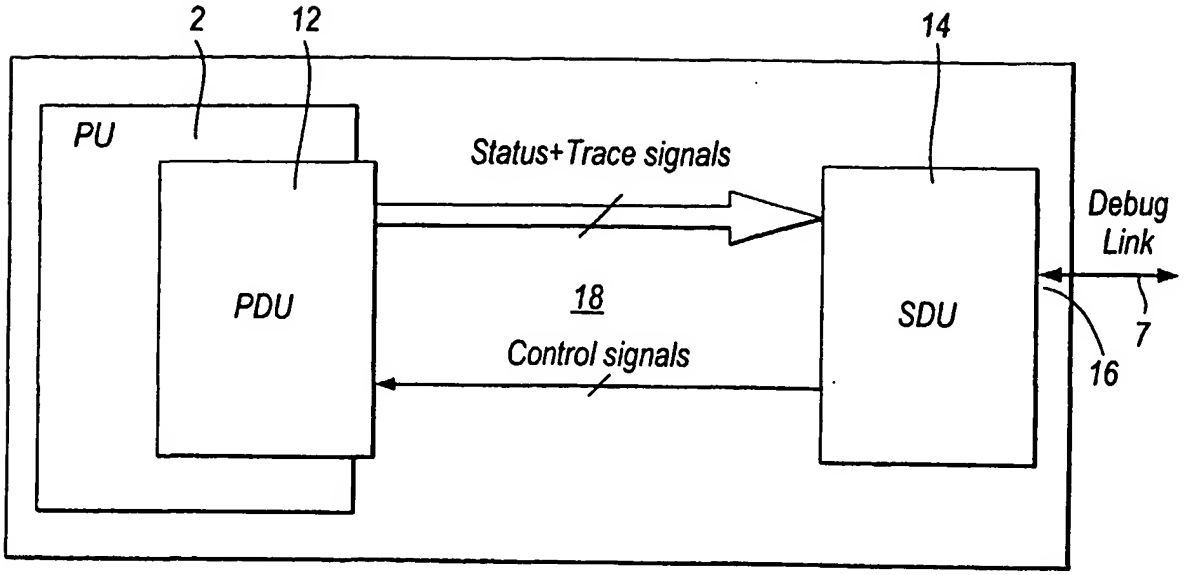

2
12
14
PU
PDU
Status+Trace signals
18
Control signals
SDU
Debug Link
7
16

Fig.2

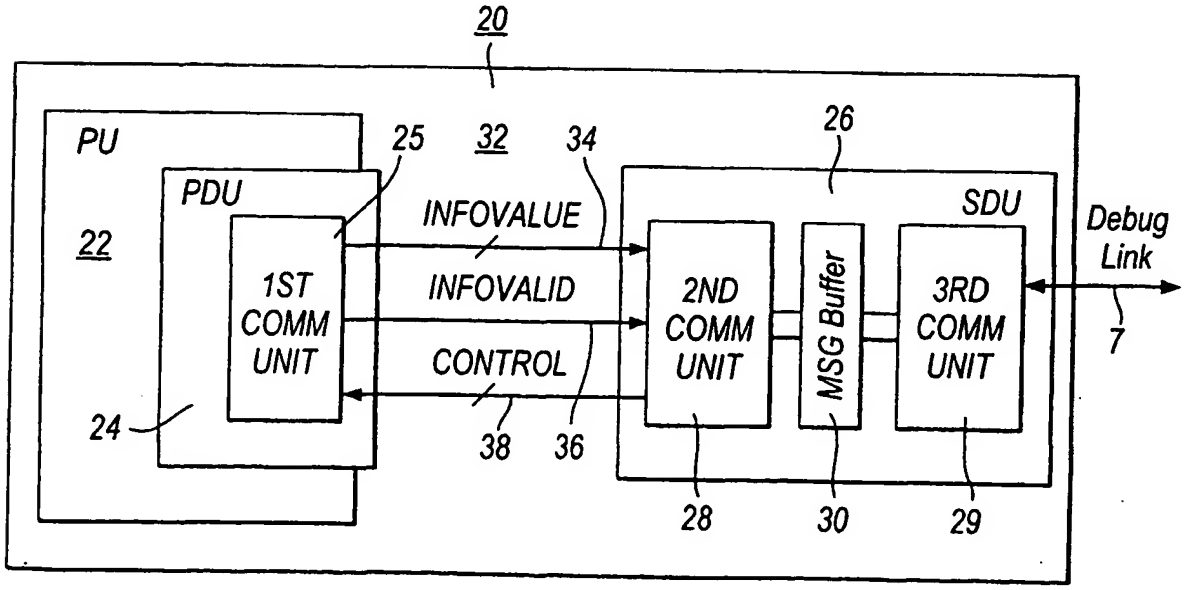

20
PU
22
PDU
1ST COMM UNIT
24
25
32
34
26
INFOVALUE
INFOVALID
CONTROL
38
36
SDU
2ND COMM UNIT
MSG Buffer
3RD COMM UNIT
28
30
29
Debug Link
7

Fig.3

DEBUG INFORMATION SIGNAL	NO. OF BITS	PURPOSE	IBW/DBW EXCEP-TION	IBW TRACE MESSAGE GEN	DBW TRACE MESSAGE GEN	DQM TRACE MESSAGE GEN	BTM
PC_VALUE	32	PC value of last retired instruction		✓	✓	✓	✓
CASN_VALUE	8	CASN value in place when last instruction was retired		✓	✓	✓	✓
DATA_VALUE	8/16/32/64	If the last retired instruction triggered a DBW event detector, contains the data value read/written			✓		
DBW_ADDRESS	32	If the last retired instruction triggered a DBW event detector contains the address of the data access performed			✓		
IBW_HITMASK	4	Bitmask to indicate which (if any) of the IBW event detectors triggered	✓	✓	(✓)		
DBW_HITMASK	4	Bitmask which indicates which (if any) of the DBW event detectors triggered	✓		✓		
PCDEST_VALUE	32	PC value of the instruction following the last retired instruction. Normally this is PC_VALUE + size of (instruction), but for branches contains the correct non-sequential address					✓
DQM_VALUE	32	If the last retired instruction triggered a DQM, contains the data value written to the DQM register				✓	

Fig.4

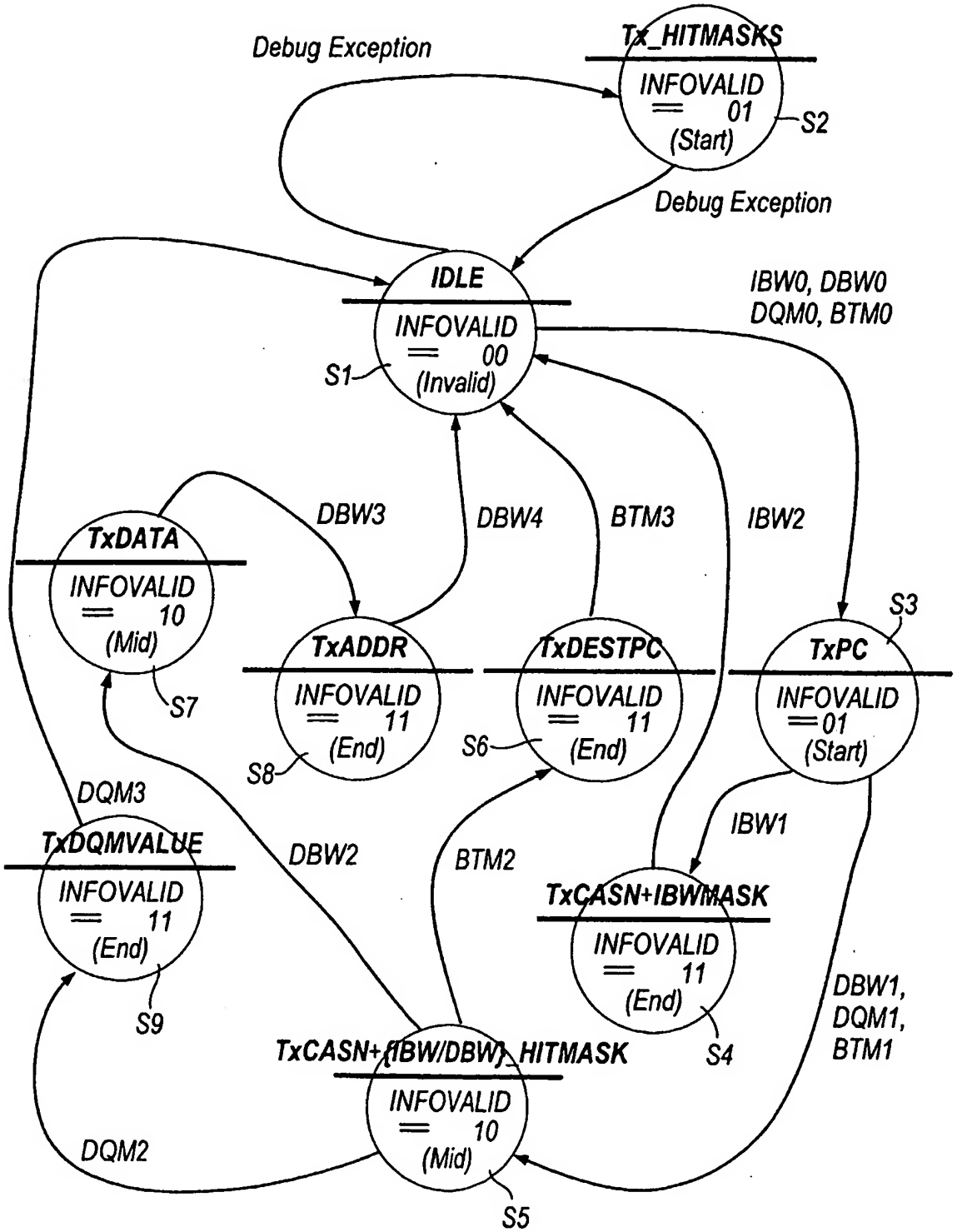

Debug Exception
Tx_HITMASKS
INFOVALID
= 01
(Start)
S2
Debug Exception
IDLE
INFOVALID
= 00
(Invalid)
S1
IBW0, DBW0
DQM0, BTM0
DBW3
DBW4
BTM3
IBW2
TxDATA
INFOVALID
= 10
(Mid)
S7
TxADDR
INFOVALID
= 11
(End)
S8
TxDESTPC
INFOVALID
= 11
(End)
S6
TxPC
INFOVALID
=01
(Start)
S3
DQM3
TxDQMVALUE
INFOVALID
= 11
(End)
S9
DBW2
BTM2
IBW1
TxCASN+IBWMASK
INFOVALID
= 11
(End)
S4
DBW1,
DQM1,
BTM1
TxCASN+{IBW/DBW}_HITMASK
INFOVALID
= 10
(Mid)
S5
DQM2

Fig.5

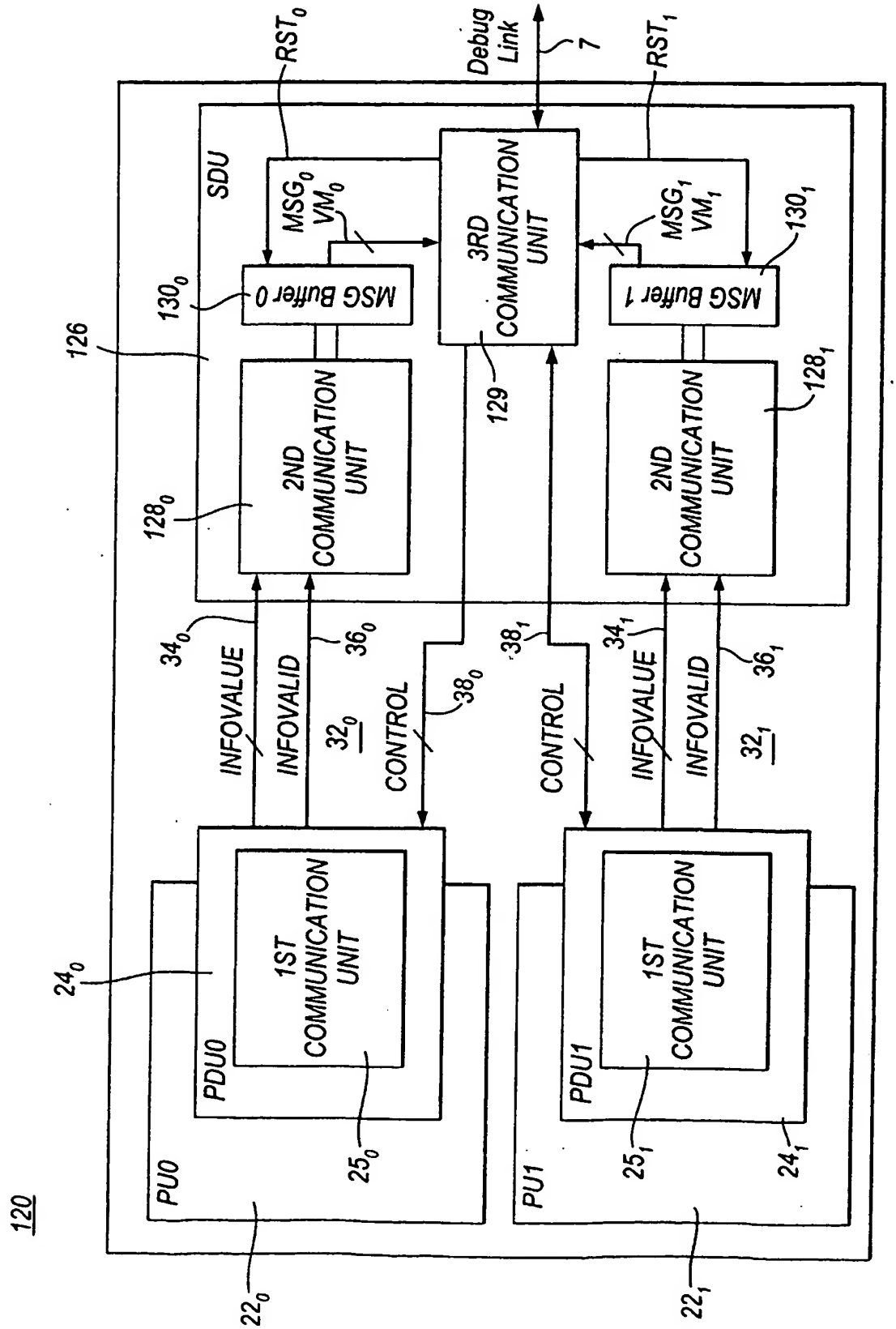

120
SDU
RST0
RST1
Debug Link
7
MSG0 VM0
MSG1 VM1
3RD COMMUNICATION UNIT
MSG Buffer 0
MSG Buffer 1
1300
1301
126
1280
1281
2ND COMMUNICATION UNIT
129
340
360
381
341
361
INFOVALUE
INFOVALID
320
CONTROL
380
PU0
PU1
PDU0
PDU1
1ST COMMUNICATION UNIT
240
250
251
241
220
221

Fig.6

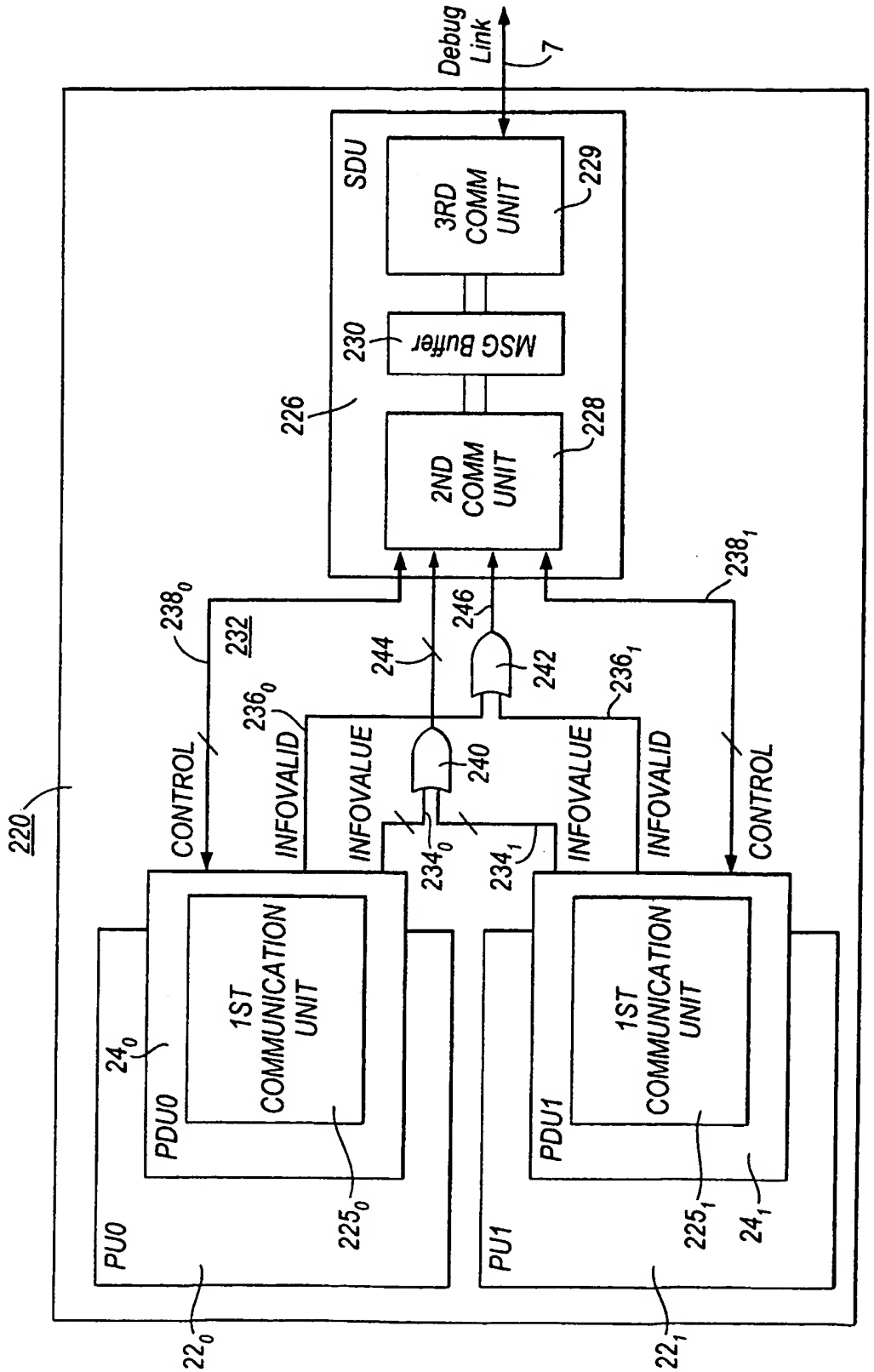

Debug Link
7
SDU
3RD COMM UNIT
229
MSG Buffer
230
2ND COMM UNIT
228
226
220
232
238$_0$
238$_1$
246
244
242
240
236$_0$
236$_1$
CONTROL
INFOVALID
INFOVALUE
INFOVALUE
INFOVALID
CONTROL
234$_0$
234$_1$
PU0
PDU0
1ST COMMUNICATION UNIT
24$_0$
225$_0$
22$_0$
PU1
PDU1
1ST COMMUNICATION UNIT
225$_1$
24$_1$
22$_1$

Fig.7